TRAC-M-TR-14-023
15 June 2014

Concentration of Risk Model (CORM) Verification and Analysis

TRADOC Analysis Center - Monterey
700 Dyer Road
Monterey, California 93943-0692

This study cost the Department of Defense approximately $40,000 expended by TRAC in Fiscal Years 13-14.
Prepared on 20140711
TRAC Project Code # 060106.

TRAC-M-TR-14-023
15 June 2014

Concentration of Risk Model (CORM) Verification and Analysis

Edward M. Masotti
Sam Buttrey

TRADOC Analysis Center - Monterey
700 Dyer Road
Monterey, California 93943-0692

PREPARED BY:

EDWARD M. MASOTTI
MAJ, IN
TRAC-MTRY

APPROVED BY:

Christopher M. Smith
LTC, FA
Director, TRAC-MTRY

This page intentionally left blank.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
15-06-2014	Technical Report	October 2013 - January 2014

4. TITLE AND SUBTITLE

Concentration of Risk Model (CORM) Verification and Analysis

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Edward M. Masotti,Sam Buttrey

5d. PROJECT NUMBER

TRAC Project Code 060106

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

TRADOC Research Analysis Center, Monterey, CA
700 Dyer Rd,
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

TRAC-M-TR-14-023

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Army Analytics Group

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited. This determination was made on 15 June 2014.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The goal of this project was to reproduce an analysis that had been done in the Department of Health Care Policy, Harvard Medical School. This project represented one of the first efforts to exercise all the capabilities of the PDE. After receiving and reproducing scripts, we were able to reproduce datasets that were very similar but were unable to verify Harvard's results. This effort identified significant issues with the development process including system system and connectivity related issues and design choices in the code. We recommend that PDE administrators consider the increasing processing power and that project managers consider improvement of the quality assurance and quality control of project deliverables.

15. SUBJECT TERMS

Risk, Concentration of Risk Management (CORM), Person-Event Data Environment (PDE)

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	Unclassified	32	MAJ Edward Masotti
Unclassified	Unclassified	Unclassified			**19b. TELEPHONE NUMBER** *(include area code)* (831) 656-6271

This page intentionally left blank.

TABLE OF CONTENTS

LIST OF FIGURES

This page intentionally left blank.

1. INTRODUCTION

The Concentration of Risk Model Verification and Analysis project is a continuation of the work conducted in 2013 which described progress and setbacks in the Study to Assess Risk and Resiliency in Soldiers (STARRS) Validation.[1] This document describes work performed in both FY 2013 and 2014.

1.1. Background

The goal of this project was to reproduce an analysis that had been done in the Department of Health Care Policy, Harvard Medical School. That analysis, done in concert with the National Institutes of Mental Health and using data from a repository at the University of Michigan, had attempted to identify soldiers at higher-than-average risk of suicide. We will use "Harvard" to mean "the set of analysts at Harvard." Harvard's work had consisted of building the data set and then producing and running a model on that data. Our task was to construct the very same data ourselves using Harvard's SAS code but using the data supplied in the Person-Data Environment(PDE).

1.1.1. Person-Event Data Environment (PDE)

This project represented one of the first effort to exercise all the capabilities of the PDE. As a result, a number of the insights we provide speak more to the usefulness and growth in capability of the PDE than to the analysis of the project itself. In this section we recap the project and describe where it finished. In Chapter 4 we list some of the obstacles that made this project less successful than it might have been. It is our hope that future analysts will benefit from being aware of some of the issues early on.

The PDE is a virtual computer environment set up in association between the Defense Manpower Data Center and the Army Analytic Group. Users connect remotely to a server inside the PDE and, once there, can access data and analysis tools that are otherwise insulated from outside. In this way the security of personally identifiable information is preserved, and Institutional Review Boards satisfied. The PDE includes an Oracle database from which our data was drawn and the SAS Enterprise Guide into which Harvard's code was imported.

1.2. Key Terminology

For the purposes of this project it is important to understand some key terms which are described below.

1.2.1. Raw Data

Raw data is defined as the manpower data as delivered by the Army through the PDE and stored in Oracle tables within the PDE.

1.2.2. Cleaned Data

Cleaned Data is defined as the output from the"data cleaning process. This is the process that ensures that all the data have plausible values, so that nobody has two different fatal incidents, that everybody is either male or female, that nobody who has earned a college degree later loses it, and so on.

1.2.3. Code

Code is defined as the SAS and R commands used to turn raw data into cleaned data and produce models.

1.2.4. Model

Model is defined as a prediction rule, expressed in SAS or R, that allows the user to predict the probability of suicide for any soldier. The model will be produced using, as input, predictors (the set of factors that held to be associated with suicide) and the response (an indicator of whether a suicide in fact occurred).

1.3. Problem Statement

The focus of this research was to verify the statistical analysis and results from the original research team's Concentration of Risk Model (CORM) using the 2004-2008 cohort data set as well as a sample from the 2010-2012 cohort. This work sought to verify the results of CORM, test the model with the most recent datasets, and make recommendations on the implementation of the model.

1.3.1. Issues for Analysis

Issue 1: Can we verify CORM?

EEA 1.1: Can we replicate Harvard's datasets?

EEA 1.2: Do we replicate Harvard's model output?

Issue 2: Can we improve PDE performance?

EEA 2.1: Can we improve the methodology for building datasets?

EEA 2.2: Can we improve model performance?

1.4. Constraints limitations, and Assumptions

Constraints limit the study team's options to conduct the study. *Limitations* are a study team's inabilities to investigate issues within the sponsor's bounds. *Assumptions* are study-specific statements that are taken as true in the absence of facts.

- Constraints
  - Complete by 30 September 2014.
  - Analysis and data manipulation will be conducted in the Person-Event Data-Environment (PDE).
- Limitations
  - We will obtain an IRB determination.
- Assumptions
  - We will be able to accurately reproduce the study team's work without complete documentation.
  - We will be able to build a cohort 2010-2012 test data set within the PDE.
  - We will receive and maintain access to the PDE.
  - We will receive the Harvard model in a timely manner.

1.5. Project Team

- **Sponsor:** TRAC-MTRY
- **Project Lead:** MAJ Edward Masotti (TRAC-MTRY)
- **NPS Faculty:**
  - Dr. Sam Buttrey

1.6. Project Methodology

The project began with four main phases. Midway through the project execution, the sponsor, Army Analytics Group (AAG), asked us to cease work and to provide documentation of the progress achieved. Below describes the original methodology as well as what we did accomplish with each phase. The first phase was focused on the verification of CORM. We

were able to reproduce the majority of the datasets that Harvard produced but did not receive an actual model. Thus, we did not achieve verification of the model. The second phase was to be focused on data development. We wanted to build a new dataset using the 2010-2012 cohort and identify performance issues with this dataset and model. This phase was not achieved. The third phase was to focus on analysis and improvement. We wanted to apply the model to the 2010-2012 dataset as well as identify performance improvements in the PDE and SAS code. This phase was partially realized as we were able to identify potential performance improvements in both the PDE and SAS code. Finally, we provided documentation of all methodologies and analyses to the sponsor in the form of this technical report. This methodology is shown in figure 1–1.

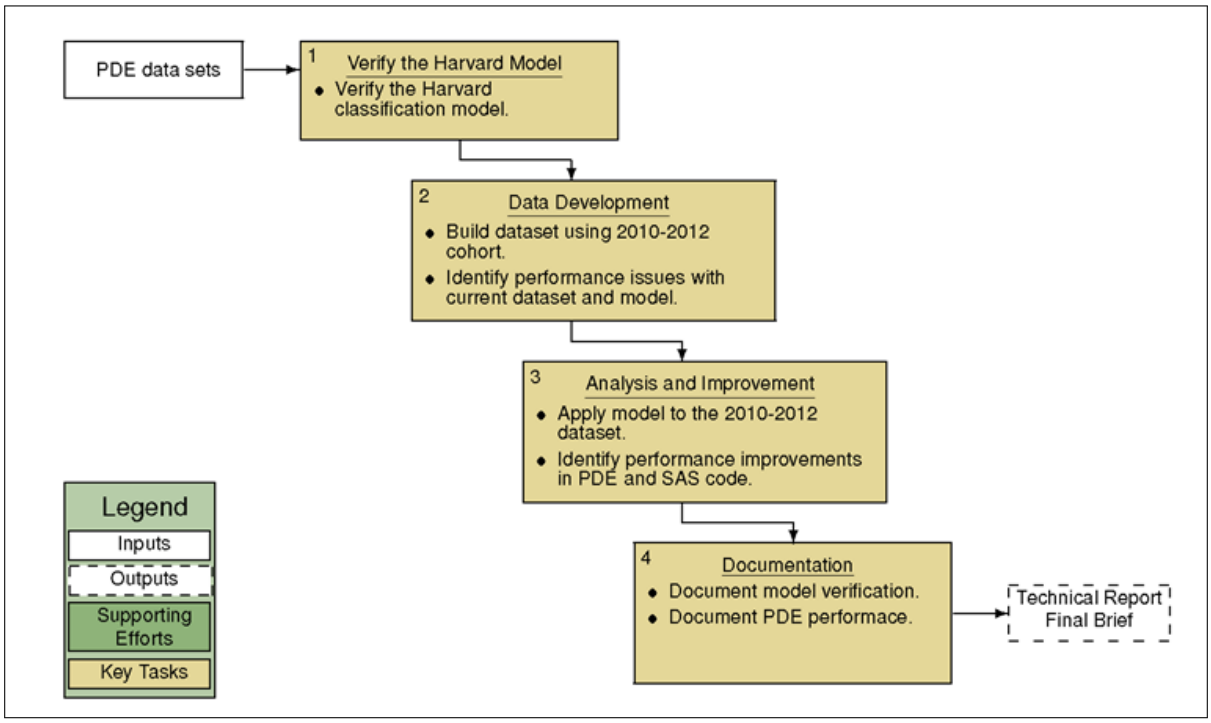

Figure 1–1: CORM methodology.

1.7. Project Timeline

We executed this project according to the following timeline.

30 OCT 13 Initial IPR.
15 NOV 13 IRB determination.
30 NOV 13 Receive Model.
15 JAN 14 IPR #2.
22 JAN 14 CORM verification complete.
31 MAR 14 Documentation complete.

1.8. Report Organization

Chapter 1 of this technical report gives the background information, the problem definition, and overview of the research methodology. Chapter 2 provides a deeper description of the methodology. Chapter 3 provides the results. Chapter 4 provides discussion of the challenges encountered in the conduct of the project. Chapter 5 provides our recommendations.

2. METHODOLOGY

This project focused on verification the Harvard model. The methodology consisted of six key components and applies equally to the source code required to build the datasets as well as the model itself. These components include receipt of the code, modification of the code to enable operation with the PDE environment, running the code in SAS within the PDE, identifying and correcting errors received, looping through running the code until no errors occur, comparing the output to the Harvard team's output, and maintenance of the of the PDE file base to ensure sufficient disk space remains to continue production of the next "round" of code. This is shown in figure 2–1.

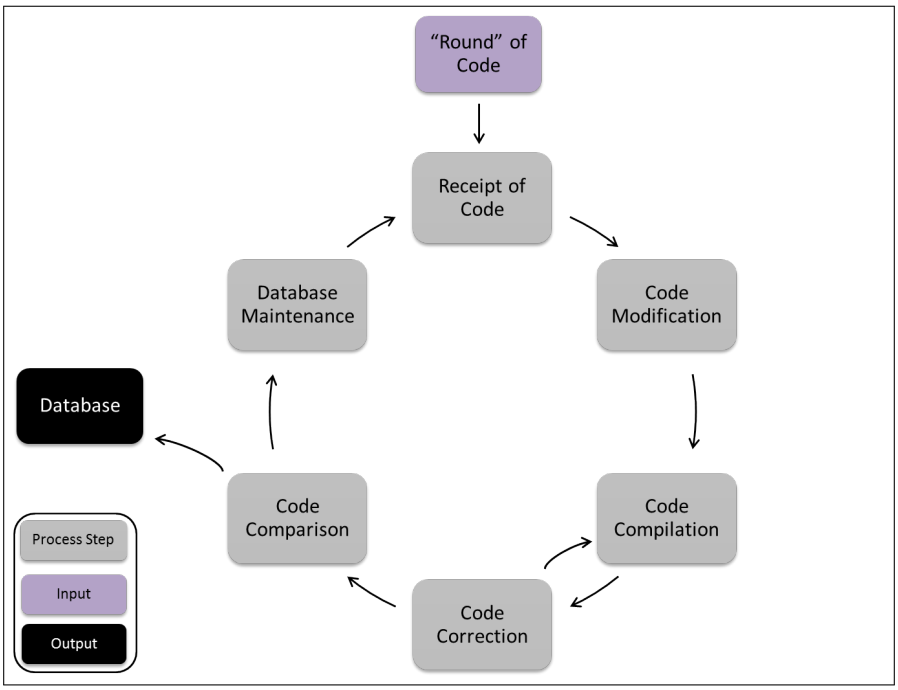

Figure 2–1: Database development methodology.

2.1. Receipt of Source Code

The Harvard team developed the CORM model in segments using SAS on the Harvard database and then delivered these segments as they were completed. The code was delivered in four pieces ("rounds") in October of 2012 and then an updated set was delivered in April 2013. That updated set was further augmented in May 2013. The code also included some limited documentation, log files (showing what happened when Harvard ran the code at their facility) and some datasets (with, for example, ICD9 codes). A fifth round was delivered in July 2013.

2.2. Code Modification

The code received cold not be immediately run within the PDE due to a number of challenges. First, Harvard's used a Linux based system to develop the code. The PDE is in the windows environment. As such, file path convention for the two systems is different. Therefore any time a file path was used in the code, modifications were required to enable operations in the windows environment. Secondly, the naming convention of variables was not consistent between Harvard's database and the database within the PDE. These challenges are further described files within the in Chapter 4.

2.3. Code Compilation

After code modification and with the realization of the file path and naming convention challenges, we attempted to break the scripts into pieces. This also led to difficulties such as issues with global variables, macros, and deletion of intermediate products. In the end, it became necessary to reproduce our own scripts to build the datasets. Using Harvard's code and databases as a baseline, we developed scripts using SAS in the PDE environment to reproduce the Harvard's output.

2.4. Code Correction

The code compilation was an iterative process in which we compiled each script one at a time until we received errors. We then corrected errors and ran the script again. We continue this process for each script until no errors occurred. When a script was complete we moved on to the next script until all scripts were complete.

2.5. Code Comparison

During the code compilation and code correction process, we compared our output to Harvard's. Often times, even when the scripts complied without errors, the output was not similar and we would again iterate through the process until the outputs were analogous. The final output of a complete round was a database that would be used as input for the next round or within the final model.

2.6. Database Maintenance

After a round was complete, it was critical to conduct maintenance within the PDE. Although improvements have been made in disk space and processor speed, at the time, disk capacity would often be approached. It was necessary to delete intermediate files that were necessary in the production of the output but were not going to be used in subsequent steps. This

produced its own set of challenges as files be deleted accidental or either by human error or without the realization that they would be used in a subsequent step (this occurred most often when it was necessary to run in another round).

3. RESULTS

3.1. Results

We completed running the original four rounds of code in October, 2013. The running of the fifth round was not completed. We are not certain how our results compare to Harvards. We expect some differences, because, realistically, we do not expect our starting data to have exactly matched theirs. Nor can we do the same specific operations on individual soldiers records that Harvard did, since the naming convention in Harvard's database and the PDE database are different. However, work in comparing log files suggests that in fact the two resulting data sets are quite similar.

As discussed earlier, we did not receive a model. Harvard did produce "source code" that they described as a model. It fell short of a model in that it was not predictive. One of the variables was the year which meant that the code could only be applied to the current dataset and could not predict future risk. It also came with comments embedded in the source code which stated they were still working on the final model. Through the completion of this project, no additional code was provided.

3.1.1. PDE Improvements

We consider this project as part of the PDE's startup cost. Had the PDE been configured at the beginning of the project as it is now, we expect that this project could have been completed in a few months. The primary improvements to the PDE during the life of the project were these.

- **Sufficient disk space:** The initial configuration of the SAS server was insufficient for the project. The current setup, at about 2TB on the main disk, is just barely adequate. Given that disk space is comparatively cheap, we urge administrators to obtain and make available as much disk space as possible.
- **Permission control:** When PDE technical staff was able to arrange for SAS Desktop sessions to continue after a disconnection, progress was hugely improved. It is important to note that the settings for the SAS Server are distinct. We thank the technical staff for their flexibility and recommend that these settings be maintained the way they are now.

4. DISCUSSION

In this section we lay out the obstacles that impeded progress on this work. The intent here is to record our experiences as an aid towards future developers in future projects. We are not intending to cast blame on collaborators, nor to exonerate ourselves for portions of the project that took unexpectedly long to perform.

We have divided obstacles into two types, although in fact these overlap. The first set of obstacles includes a number that have now been remedied; again, we record these for the benefit of future developers in other environments. These are system- and connectivity-related issues and refer mostly to the inadequacies and vicissitudes of hardware and networks. The second set of obstacles concerns certain design choices in the code. We believe that this code was developed in a piecemeal manner in order for its users to answer a particular question one time. It was not the result of a rigorous software development effort designed to produce a re-usable product, and we recognize that. Nonetheless some of these design choices made it difficult for us to work on the code and the data, and so we detail those issues in that section.

4.1. System Issues

In this section we detail the system-related obstacles that future researchers should be aware of. These can be roughly broken into categories representing connectivity issues, on the one hand, and disk space and other system resources, on the other. Of particular import is the behavior of SAS on abnormal termination, so we discuss that briefly as well.

4.1.1. Connectivity issues

The nature of the PDE is such that a lot of computers need to be in communication with one another more or less continuously during processing. These computers include:

- The clients computer (at which the analyst sits physically);
- The SAS Desktop (which serves as the virtual home for the analyst in the PDE, and from which the analyst can run the SAS Enterprise Guide);
- The SAS Server (on which SAS itself actually runs);
- The Oracle Server (where the Oracle database is maintained and served).

While we do not have much information about the internal construction of the PDE, it certainly makes sense to maintain a mental model that looks like figure 4–1:

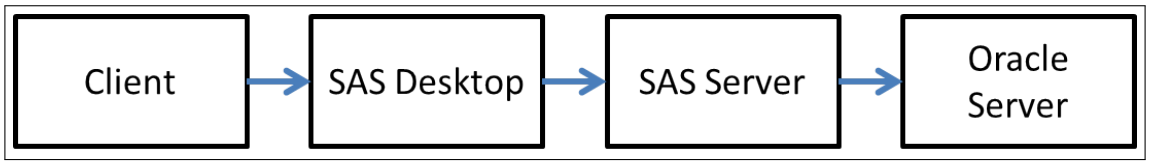

Figure 4–1: Mental Model of PDE

When our work on the project began, a failure of any component (either a computer or a link) was, we believe, enough to cause the SAS system to stop working. The connection between the client machine and the SAS Desktop was particularly troublesome. That connection, made via Citrix software, was very fragile, and, again, at the beginning of the project, any disconnection caused SAS to terminate abnormally. (See "SAS WORK and What Happens On Termination," below.) A restart of the client computer, as another example, would terminate the entire job, and these restarts were distressingly frequent as a result of, for example, software updates at the administrative level.

After some investigation, PDE technical staff were able to find and alter certain timeout settings in such a way as to keep SAS Desktop sessions open for some length of time even when the client connection was temporarily lost. This was one of two main changes that made it possible for this work ever to be done. (The second was the addition of a large amount of disk space; see "Memory and Processing bottlenecks" below.)

Of course, each of the computers and connections in our mental picture is subject to planned events like shutdowns for maintenance and unplanned events like system crashes. Even if the frequency of any one event is low, the chances of even one interruption during a three-day job is not always insignificant. We developed a number of strategies to protect against the delays caused by interruption, although in retrospect none of them was particularly successful. We note that the connection from the client to the SAS desktop was very much the weakest link in this chain. We did observe failures of all sorts, but, as we said in the previous paragraph, real progress was very difficult until PDE staff was able to alter timeout settings so that SAS Desktop sessions could remain working even when clients disconnected.

Without our having spent much time on quantifying this, our intuition is that, before these timeout settings were altered, about 80% of stoppages were the result of the client disconnecting from the SAS Desktop; perhaps another 10% were the result of the client machine crashing; and most of the rest appeared to be associated with issues on the SAS Desktop itself. We did, however, see occasional evidence of the SAS Server crashing (or at least disconnecting from the SAS desktop) and of the Oracle server crashing or being inaccessible. Again, because of the large number of complex entities and connections, a certain crash rate is to be expected; the issue was, for us, the general inability to recover.

It is worth noting that there is a set of timeout settings for access to the SAS Server itself that appear to be separate from those for the SAS Desktop. One approach we used was to try to

run SAS jobs in "native" SAS directly on the SAS Server, rather than use Enterprise Guide from the SAS Desktop. This approach required that the Desktop's own security settings be properly adjusted.

4.1.2. Memory and processing bottlenecks

A second set of issues arose when the resources of the PDE were inadequate to handle the demands of the code. (In part this may have been due to inefficiencies in the code, but there is no way to get around the fact that this is a large problem.) For months the available disk space on the SAS Server was measured in the hundreds of GB, but we learned in August 2013 that Harvard's server required 2 TB of data in order for their scripts to run. PDE technical staff were able to secure and install the required disk space, but only, of course, after this requirement was made known to them. Disk space tended to be used up by intermediate products that were not removed because SAS terminated abnormally (see section B.1.c). This created another burden on the PDE technical staff; that of policing the disk drive in search of large files that could be safely removed. When the disk on the SAS Server filled, SAS terminated abnormally.

In addition to the lack of disk space we suspect that the SAS Desktop's stability could be endangered by the presence of multiple users. At least, we observed a number of crashes that did not appear to be attributable to other causes, or to bad luck. As the PDE grows the number of machines dedicated to handling users should probably grow as well.

4.1.3. SAS Work and What Happens on Termination

Intermediate products computed by SAS are stored in a directory named WORK which resides on the SAS Server. These products are regular disk files with extension *sas7bdat* and they persist until either they are explicitly deleted inside the SAS code, or SAS terminates normally. When SAS terminates, all the members of the WORK directory are deleted from the disk. If SAS terminates abnormally, items in WORK that have been completed will normally be complete and usable in a future session, although we were wary of using SAS items recovered after a crash. (Certainly an item that is in use when SAS terminates abnormally will be unusable, because it will be either corrupt or incomplete.) Conversely if SAS terminates abnormally, items in WORK that are preserved in this way need to be explicitly removed in order to free up their disk space.

4.1.4. Our strategy for recovering from a failure

Before it became clear that the PDE lacked sufficient disk space to run the Harvard code in the form in which it was presented, we attempted to run code through a makeshift strategy of breaking scripts into pieces and running them bit by bit. This approach turned out to raise problems of its own; see section 4.2.4, Breaking Scripts into Pieces, and section 4.2.4, "Removal of Intermediate Products."

4.2. Code Issues

In this section we describe some of the issues that required attention or correction before Harvard code could be used in the PDE.

It must be said that Harvard's code was clearly not the product of a software development process designed to produce streamlined, well-tested code. Instead it had the hallmarks of a set of ad hoc programs put together in order to solve a particular problem one time. This is understandable given the nature of the project, but it made the code's re-use difficult. We have all had the experience of having tested, verified code fail for reasons that are difficult to detect; this is all the more common in someone else's one-time code.

The code was only sporadically documented. (Documentation on the outside, describing the flow from one program to the next, was somewhat better.) Certain coding standards that we have come to expect (the use of meaningful variable names that are not re-used from one task to a different one) were not always met. Again, we understand that code is often developed under time pressure and the rewards for comprehensive documentation are small. Nonetheless this is not the sort of code we would present to clients.

4.2.1. Mismatched file and library names

Tables in Harvard's data center often had slightly different names than the corresponding tables in our Oracle database. References to tables with different names had to be found and changed. In principle this might have been done automatically after constructing a table with one row per table and two columns giving the different names, but in practice we did this by hand.

It is also the case that Harvard's own internal table- and library-naming conventions were inconsistent. For example the final script from round 1, script 2 is named final1 and stored in a library named linuxdrv. In script 5 this is read in from a library named new1, and, separately from new. We learned that these libraries were maintained by different analysts. It might be worth noting that, also in round 1 script 5, another dataset named final1 is created. This is not the same as the earlier dataset by that name.

4.2.2. Changing the ID name

The simplest change we have to make to the code was to convert all references to Harvard's ID (which they called PID_CHPPM) to the ID in the PDE (called PID_PDE). For most scripts, a simple global change was all that was necessary. However, we did encounter several issues with IDs. First, in a number of places the Harvard code specified particular IDs, presumably to resolve issues with individual records. Because we lacked the ability to convert CHPPMs to PIDs, we ignored all of these references.

Second, in a number of cases the code creates additional ID columns (in, for example,

dcips_injury_afmets_death_agregation). These new columns were given width 9 in Harvard's code, whereas we require that they have width 12. These references had to be found and changed.

Third, in a few instances the SAS code would refer to the column names directly, which caused problems with case-sensitivity. SAS variable names are not case-sensitive, so we got into the habit of making a global, case-sensitive conversion of all instances of PID_CHPPM (or pid_chppm, or whatever; we saw plenty of each) to pid_pde in every script. But, for example, on line 1798 of allsx_cpt_v7_linux_3 we see this macro code: %let var = %sysfunc(varname(&dsid, &i)); %if &var ne yearmonth & &var ne PID_PDE &then Here the value contained in var is being compared to the strings yearmonth and PID_PDE. This comparison is a case-sensitive one, so if in this case an instance of PID_CHPPM (upper-case) was converted to pid_pde (lower-case), this code will fail.

In short, even the act of converting one ID name to another could cause problems.

4.2.3. Code That Did Not Work

Direct Sorting of Oracle data

In some cases code that we were provided did not run in our environment. One type of difference arose from the different ways that data was being accessed by the SAS engine Harvard used an ODBC connection, while we used the SAS/ACCESS interface to Oracle. If tmds were the name of an ODBC connection, then Harvard's code would use that library and execute a command like this (this example comes from make_tmds_constructs_1.sas): Proc sort data=tmds.tmds_pem_inpt_dt (other clauses here)

We presume that this call was successful for Harvard, but it does not appear to be valid in the case where tmds is an Oracle connection. Instead, we needed to read the data into SAS first, and sort in a separate step, like this: Data holder; set tmds.tmds_pem_inpt_dt; run; Proc sort data=holder (other clauses here)

Transposition without Prefix

In some cases (e.g. make_tmds_constructs_1.sas) the Harvard code used proc transpose to transpose a matrix. The PREFIX= option specifies a prefix to be added to the front of the names of the newly constructed columns. Without a prefix, SAS would, in some cases, produce columns names that are invalid (because, for example, they start with a numeric character. Therefore SASs default is to add an underscore as the prefix character. It was discovered, with Harvard's help, that SAS Enterprise Guide does not honor this default (although the documentation seems to suggest that it is supposed to). So those transpose commands that use var and group clauses on a numeric column, and do not specify an explicit PREFIX, will fail under SAS Enterprise Guide.

This causes particular trouble in script 4 of round 2 (alldx_merge_master_4). We observed a number of errors running this script. In an earlier document we attributed a those errors to referring to diagnoses for which there were no soldiers recorded. It is now established that these errors were a result of this bug in SAS Enterprise Guide.

4.2.4. Breaking Scripts into Pieces

During the time that we found ourselves unable to run jobs to their completion, because of connection or resource issues, we tried to break the scripts into pieces and run the pieces one at a time. This approach, too, ran into difficulties. These difficulties included the use of global variables, macros, and deletion of intermediate products.

Global variables

Some scripts used global variables. These are variables that do not belong to a SAS data set. They can be set or retrieved anywhere in the code. As an example of their use, imagine trying to construct a dataset with one row for each individual and one column for each separate diagnosis for that individual. It would be useful to know the maximum number of diagnoses that any individual had, in order to dimension the resulting data set. So one way to accomplish this would be to read all diagnoses, accumulating counts by ID, and saving the maximum number of diagnoses to a global variable.

The issue is that when running code in pieces any global variable that will be needed in a particular piece has to be identified and set before the piece can be started. A more general solution might have been to write these variables to a small text file or SAS data set for more permanent storage between runs.

Use of macros

A SAS macro is a sub-function that allows the re-use of code in different contexts. Macros can be very useful and we will not argue for their elimination. However, SAS Macros are written in a modified form of the language that makes reading and understanding macro code difficult. We believe that long complicated macros deserve particularly strong documentation.

If a section of code needs to be run several times, a macro is a natural choice. However, it is difficult to run only a piece of a macro, which is best seen as an indivisible unit. So a script consisting of a macro definition followed by a single invocation confers no advantage in efficiency over regular code and suffers from being very difficult to divide into pieces. We encourage developers to avoid this.

As a particularly egregious example of macro misuse, consider the active_duty macro in the check_trans_v26_linux_2 script. This is defined twice, using different definitions. Therefore

when the reader encounters a call to a macro of that name he or she cannot uniquely determine the contents of the macro being called; the macro being called depends on the callers location in the program. This reduces readability enormously.

Removal of intermediate products

In a number of instances, the SAS code explicitly removes intermediate products. This is a good practice when disk space is at a premium, but a bad one when crashes are common, since if all intermediate products up through a particular point can be preserved, the program can be re-started at that point with no loss. We do not fault Harvard for their choices in this matter, but, seemingly inevitably, we found ourselves both running out of disk space and being unable to restart at intermediate points.

Copying results across file systems

Because of insufficient disk space, we would often attempt to run a piece of the SAS code, copy intermediate products over to another file system on the network, delete the product from the SAS server, and then resume. While this makes sense in principle, it is painful in execution. These disk copy operations often took several orders of magnitude longer than would an equivalent copy within a file system and, as we have said, the danger of a crash during any extended operation is non-zero. Still, when disk space was at a premium something had to be done.

5. RECOMMENDATIONS

5.1. Recommendations

We recommend that PDE administrators consider the increasing processing power. We suspect, but cannot prove, that some crashes were caused by contention among users for resources, either those of SAS, or Oracle, or of the network. As the number of PDE users grows, so too will this contention. We recommend that PDE administrators continue to have the environment grow in terms of the processing resources available. Perhaps parallel computing architectures can be brought to bear for the big jobs we expect to see in the future.

We also recommend that project managers consider improvement of the quality assurance and quality control of project deliverables. The code given to us by Harvard was inadequate for the task. We recommend that quality control be performed on the code and the documentation as part of any contract. If code cannot run on our system because we use SAS/ACCESS and Harvard uses ODBC, that is understandable. If code cannot be run because Harvard's own internal naming system is inconsistent, that is less so.

A. THE ROUNDS OF CODE

Harvard supplied five sets ("rounds") of code. The documentation is not very forthcoming on which round undertakes which task, but in this section we give our interpretation of what the code is trying to accomplish. In most cases we rely on the comments in the code to deduce the intent.

A.1. ROUND 1

This is the primary "data cleaning" module. The jobs in this round try to form a consistent and complete data set. This takes two forms. First, many jobs apply corrections to the data to try to ensure consistency. For example, the "Educ_correct_linux_3" script apparently tries to correct records where a soldier's education level is lower in a later month than in an earlier one. Second, the scripts look to different sources to find the data in question. For example, "Gender_new_6" extracts the gender from each of 23 tables and uses whichever non-missing value it finds.

Naming convention: Many of the scripts have dates in their name; we presume this is intended for version control. A trailing number indicates the order in which the scripts are to be run, so Age_new101411_8 is run immediately after Race_ethnicity11182011_7.

A.2. ROUND 2

Round 2 creates the data sets with inpatient and outpatient diagnoses from the TMDS data base. This takes in the Harvard-supplied data set icd9_091112.csv, which gives ICD codes and corresponding diagnoses.

A.3. ROUND 3

This round cleans MOSs and assigns to them "factors" associated with the ONET civilian job classification system. (Presumably this is the scheme described at http://www.occupationalinfo.org/onet/.) The mos_readme_100912.docx and mos_variables.docx files gives somewhat more information than is often presented for other jobs.

A.4. ROUND 4

The fourth round of code is intended to add, to the overall manpower data set, medication data.

A.5. ROUND 5

This round creates, and merges with the master file, data sets with suicide (and related) events, in- and out-patient hospitalizations, and injuries.

GLOSSARY

AAG	Army Analytics Group
Cleaned Data	The output from the"data cleaning process.
Code	SAS and R commands used to turn raw data into cleaned data and produce models.
CORM	Concentration of Risk Model
Model	A prediction rule, expressed in SAS or R, that allows the user to predict the probability of suicide for any soldier.
PDE	Person-Event Data Environment
Raw Data	The manpower data as delivered by the Army through the PDE and stored in Oracle tables within the PDE.
STARRS	Study to Assess Risk and Resiliency in Soldiers
TRAC	Training and Doctrine Command Analysis Center

This page intentionally left blank.

REFERENCES

[1] Thomas M Deveans and Sam Buttrey. Study to Assess Risk and Resiliency in Soldiers (STARRS) Validation. Technical report, 2013.